Departamento de Engenharia de Computação e Automação Industrial
Faculdade de Engenharia Elétrica e de Computação
Universidade Estadual de Campinas

Relatório Técnico
Technical Report
DCA-003/04

Desenvolvimento de uma Interface entre MTK e Spaceball

Marcel Abrantes

Orientadora:
Profa. Wu, Shin - Ting

State University of Campinas)
FEEC (School of Electrical and Computer Engineering)
DCA (Dept. of Computer Engineering and Industrial Automation)
{ra003168,ting}@dca.fee.unicamp.br[2ex]

10 de Maio de 2003

Abstract

Apresentamos neste relatório final de Projeto de Pesquisa, financiado pela Fapesp e realizado durante o período de maio/2002 a abril/2003 (Processo n^o 02/01161-3, nível de Iniciação Científica), o desenvolvimento de uma interface para dispositivos 3D não convencionais no contexto de uma biblioteca de manipulação de objetos, o MTK, detalhando cada passo deste processo, desde o projeto inicial até a implementação e a fase de testes para depuração. Mostramos também uma reestruturação desta biblioteca cuja finalidade é a adequação às novas necessidades do projeto, visando também uma melhor organização e maior facilidade para manutenções e futuras implementações. Além disto, com a finalização da interface com o *Spaceball*, iniciamos uma fase de testes comparativos a fim de avaliar a eficiência deste dispositivo frente aos modelos convencionais em manipulações 3D.

1 Objetivo e Motivação

Com o grande avanço da tecnologia na área de computação gráfica, novas ferramentas e dispositivos surgem, permitindo aos seus usuários interagir com os sistemas computacionais de forma cada vez mais próxima da realidade (virtual).

Dentro deste contexto está o *Spacetek Spaceball*, um dispositivo de entrada que possui seis graus de liberdade, três para deslocamento e três para rotação. É uma excelente ferramenta para, através de simples pressões sobre uma esfera de controle, manipular de forma contínua os objetos 3D projetados numa tela 2D de uma estação de trabalho convencional. Em conjunto com os dispositivos de entrada convencionais, o teclado e o *mouse*, pode-se ainda, por exemplo, controlar simultâneamente a posição e o centro de interesse de uma câmera [10]. Provendo uma boa realimentação visual sobre os efeitos das ações dos usuários sobre os objetos, consegue-se aumentar a sensação de tridimensionalidade, a flexibilidade e a interatividade numa sessão de trabalho.

Esta tendência motivou o projeto do MTK (**M***anipulation* **T***ool* **K***it*). MTK é uma biblioteca que permite o desenvolvimento de aplicativos gráficos interativos 3D com uma proposta de arquitetura que suporta não só os dispositivos 2D convencionais como os não-convencionais. Esta biblioteca está sendo desenvolvida pelo Grupo de Computação de Imagens da Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campianas. Este projeto se iniciou em 1995 no contexto de um projeto de pesquisa financiado pela Fapesp (Processo n^o 1996/0962-0). Algumas das funcionalidades presentes no MTK podem ser citadas: *draggers*, controles para câmeras, para luzes, seleção incluindo a seleção 3D e apontadores 3D com movimentos livres e restritos. *Draggers* e apontadores 3D tem como finalidade proporcionar uma maneira mais intuitiva para menipular objetos. Funções de movimento restrito às superfícies e selecionar objetos em 3D foram desenvolvidas recentemente [20]. Mais detalhes são apresentados na seção 4.

Buscamos, neste projeto, articular a integração do *Spacetek Spaceball* 3003 com a biblioteca MTK, possibilitando, desta forma, uma análise futura do impacto destes dispositivos na eficiência e no rendimento das atividades de um usuário no contexto da manipulação e da interação com ambientes gráficos tridimensionais. Sendo assim, pretendemos também conduzir uma etapa de avaliação do modelo proposto de forma a analisar sua eficiência e flexibilidade frente a dispositivos convencionais, no caso o *mouse*.

Durante o desenvolvimento do projeto pudemos compreender mel-

hor o papel e o funcionamento do MTK, adquirindo confiança suficiente para que mudanças mais significativas fossem introduzidas ao mesmo. Aliada a isto, a frequente utilização da linguagem C++ e do pacote gráfico OpenGL nos concedeu experiência e amadurecimento de idéias, conduzindo o projeto a uma fase de reestruturação. Algumas das estruturas existentes no MTK foram sujeitas a adaptações visando uma melhor organização das mesmas e preparando a biblioteca para receber os novos recursos previstos no plano de trabalho, como a integração do *Spaceball* com o Cursor 3D e com o MTK em geral. A reestruturação será descrita na seção 6.2.

Será apresentada também a estrutura concretizada especialmente para o suporte de dispositivos, desenvolvida com base na reestruturação efetuada. Esta estrutura, brevemente descrita em nosso trabalho anterior [21], encontra-se documentada em detalhes na seção 7.

Entraremos em detalhes também no aspecto dos testes e e avaliação desta nova estrutura, apresentando alguns problemas encontrados durante a implementação da mesma e a soluções adotadas para contorná-los 8. Em seguida, serão apresentados testes de depuração da interface obtida e das funcionalidades desenvolvidas nesta interface.

Não podemos deixar de mencionar a reestruturação da antiga página do MTK que agora pode ir ao ar incluindo a última versão desta biblioteca e contando também com uma seção de documentação e publicações [19].

2 Plano de trabalho

O plano de trabalho seguido para e execução do projeto está especificado abaixo, assim como o cronograma para a realização do mesmo.

1. Estudo da linguagem C++;
2. Familiarização com MTK;
3. Estudo do *Spaceball*;
4. Projetar uma interface;
5. Implementar a interface;
6. Reestruturação das classes do MTK;
7. Testes e depuração;
8. Projetar funcionalidades para o MTK;
9. Integração ao MTK e Cursor3D;
10. Desenvolver um aplicativo;

11. Análise e Comparação de desempenho entre os dispositivos de entrada 2D e dispositivos não convencionais (3D) para manipulação 3D.;
12. Elaboração da documentação.

Ativ.	Mai–Jun	Jul–Ago	Set–Out	Nov–Dez	Jan–Fev	Mar–Abr
1	X					
2	X					
3	X	X				
4	X	X				
5		X	X	X	X	
6				X	X	
7					X	
8					X	X
9					X	X
10						X
11						X
12			X			X

3 Trabalhos Correlatos

Nesta seção podemos citar idéias de algumas fontes de referência relacionadas ao tópico deste projeto, as quais servem como consulta e inspiração. São diferentes paradigmas que nos auxiliam nas decisões a serem tomadas, indicando-nos caminhos que possibilitam contruir um projeto plausível.

Interfaces hápticas : Háptica [5] é um estudo de como aproximar os movimentos do sentido humano à interação com computadores. Utiliza-se para isto dispositivos manipuladores que interagem com músculos e tendões humanos gerando entradas para um sistema. É uma área muito pequisada nos Estados Unidos, especialmente na Northwestern University. Apesar de ser uma abordagem em interfaces de manipulação bastante diferente da utilizada em nosso trabalho ela merece ser mencionada.

Immersion : A *Immersion Corporation* é uma empresa que atua na área de pesquisa de tecnologias virtuais, trazendo produtos inovadores como luvas e outros tipos de equipamentos para interação em ambientes 3D [4]. São paradigmas inovadores, que devem ser levados em consideração na contexto de um projeto de interação e manipulação de objetos 3D.

3Dconnexion : A *3Dconnexion* é outra empresa que busca soluções para interação 3D. Atualmente prossegue com o desenvolvimento

do *Spaceball* e introduz novas linhas diferenciadas de dispositivos, como o *Spacemouse*. Em [1] são apresentados alguns artigos interessantes, contendo desde estudos da eficácia destes dispositivos na navegação em ambientes tridimensionais [3] até a apresentação de novos conceitos de interação e utilização destes dispositivos [2].

Pacotes Gráficos : Alguns dos pacotes gráficos mais modernos para síntese de imagens, como o *Discreet 3D Studio Max* [6], o *AliasWavefront Maya* [7] ou o *Newtek Lightwave* [8] oferecem interfaces extremamente eficientes e amigáveis para manipulação de objetos para composição de modelos e cenas 3D. São ferramentas utilizadas há tempos por muitos dos profissionais da área de computação as quais passaram por diversos processos evolutivos, devendo servir assim como base para o desenvolvimento de novas interfaces.

Controle de Camera : Russell Turner, Francis Balaguer, Enrico Gobbettie Daniel Thalmann apresentam um trabalho que descreve um modelamento matemático para movimentos naturais de uma câmera a partir de dispositivos de entrada 3D [9]. Bastante interessante, porém um pouco divergente em relação aos objetivos deste nosso trabalho.

4 MTK - *Manipulation Tool Kit*

O MTK é uma biblioteca que provê recursos para o desenvolvimento de aplicativos na manipulação e visualização de cenas e objetos tridimensionais. Esta biblioteca é composta por diferentes classes, cada uma responsável por um conjunto bem definido de funções, formando uma estrutura encapsulada que garante uma maior flexibilidade e facilidade para o ambiente de desenvolvimento cooperativo.

Devido à diversidade na forma de tratamento dos eventos pelos sistemas de janelas e os servidores associados, foi desenvolvido por Mesquita uma nova classe abstrata no MTK, mtkxWindow [16]. Foram ainda implementadas duas classes concretas de interface: mtkxGlutWindow e mtkxGtkWindow, conforme ilustra a Figura 1. A classe mtkxWindow também foi projetada para permitir a implementação da classe mtkCursor que pode ter como seu controlador um dispositivo de entrada não convencional (por exemplo, *spaceball*) ou um dispositivo de entrada convencional (como *mouse*). Porém, ao trabalhar com o *Spaceball 3003*, a falta de *drivers* para este dispositivo impediram naquele momento o desenvolvimento de uma interface com dispositivos de entrada não convencionais.

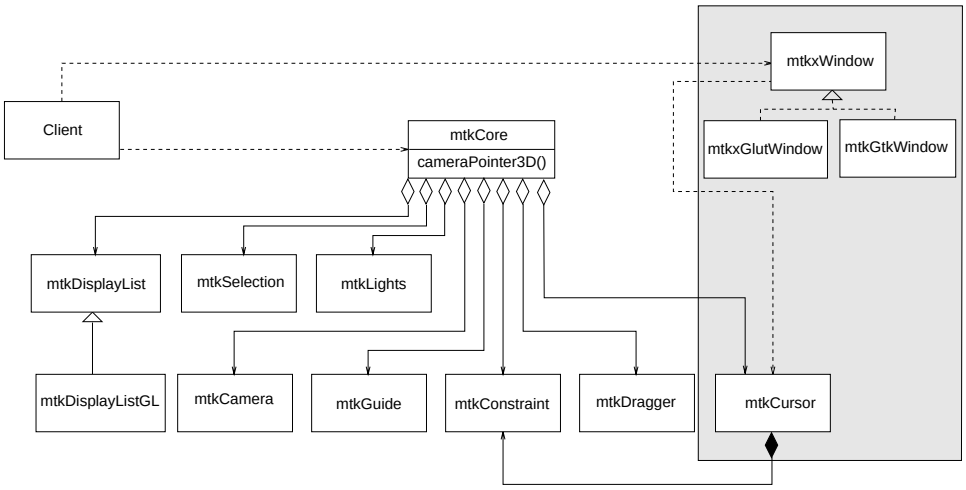

Figure 1: Diagrama de Classes do MTK.

Neste projeto tivemos, de certa forma, finalidade de dar prosseguimento aos objetivos estabelecidos quanto ao suporte a dispositivos não convencionais no contexto do MTK. Com isso, os conceitos e implementações deste projeto no que se refere ao desenvolvimento de uma interface para o *Spaceball* foram estritamente relacionados à extensão implementada por Mesquita.

5 Spacetek Spaceball 3003

O *Spacetek Spaceball* 3003 é um dispositivo de entrada que permite movimentos de translação e rotação em três eixos perpendiculares. Estes movimentos possuem uma alta resolução, o que permite uma alta precisão nos movimentos. Além disto, possui também três botões, um de uso geral no lado esquerdo, outro no direito e um botão de *Reset* também no lado direito.

Ele se comunica através de uma porta serial do padrão RS-232, disponível na maioria dos equipamentos comercializados, e vem acompanhado de um *software* proprietário denominado *SpaceWare*, que permite a integração com alguns *softwares*, incluindo também uma fonte de alimentação necessária para algumas plataformas.

Depois de alguns estudos sobre o comportamento do *Spaceball* e com base em algumas informações obtidas da biblioteca *LibSball* [11] ficou evidente o padrão de comunicação deste dispositivo. São trocados pacotes bem definidos de informação, cada um relacionado a uma ação ou funcionalidade do dispositivo. Estes pacotes têm o *byte* como unidade mínima de informação, sendo que alguns podem ser lidos na

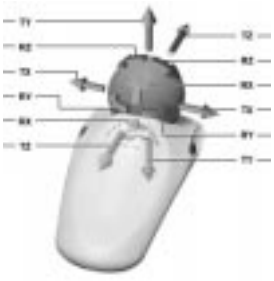

Figure 2: *Spaceball* 3003.

forma de caracteres no padrão ASCII identificando uma propriedade específica, e outros devem ser interpretados como um valor numérico. O primeiro *byte* é composto por um caractere ASCII que identifica o pacote, por exemplo M, que se refere a um pacote de movimento (*movement*). Este caractere é sempre maiúsculo quando o pacote é enviado pelo dispositivo e deve ser minúsculo quando se pretende enviar uma requisição ao mesmo para uma determinada informação. O restante é formado por dados num formato padrão para cada tipo de pacote. Por isso, cada tipo de pacote tem um tamanho fixo definido. Existem alguns caracteres especiais, como o *Carriage Return* (Hexadecimal 0D) que indica o fim de um pacote e o '^' (Hexadecimal 5E) que é utilizado para indicar que o próximo caractere deve ser tratado, que teve ser alterado para não prejudicar a comunicação já que normalmente é utilizado em fins específicos da porta serial. Informações mais detalhadas sobre o protocolo de comunicação do *Spaceball* 3003 podem ser encontradas no Apêndice A.

5.1 Interface com o Spaceball

Para se iniciar o projeto de interface, precisamos primeiramente avaliar alguns tipos de implementações já existentes as quais possuem recursos que as permitem comunicar com dispositivos não convencionais. No nosso caso, foram encontrados alguns paradigmas, compostos por aplicativos, bibliotecas ou outros recursos específicos os quais sugerem diferentes maneiras de interagir com dispositivos de entrada 3D, em especial com o *Spaceball*. A dificuldade que tivemos na localização destes paradigmas se deve em grande parte pela restrita utilização deste grupo de dispositivos.

A princípio, deve-se citar o *Spaceware*, *software* proprietário que acompanha o *Spacetek Spaceball* 3003, o qual foi a base para estudos. Apesar de não ter o seu código fonte acessível, este *software* mostrou-se de relativa importância pois, além de permitir a integração com

o dispositivo através do uso de alguns programa-exemplos que acompanham o pacote, nos apresentou o primeiro sinal de compatibilidade com o ambiente de janelas X, visto que estes mesmos exemplos são executados sobre este ambiente no UNIX.

A partir deste ponto, pôde-se dar início ao projeto e à busca de soluções para o mesmo. Os meios mais concretos e coerentes encontrados foram o recurso Xinput do próprio ambiente de janelas X e também a biblioteca LibSball, os quais são apresentados a seguir.

5.1.1 XInput

Algumas pesquisas apontavam uma possível integração entre o *Spaceball* e o ambiente de janelas X, através de um recurso denominado *Xinput* [12]. Este recurso é uma extensão do X que permite o suporte de dispositivos de entrada, além dos tradicionais *mouse* e teclado, para este ambiente.

Apesar de haver uma implementação disponível para o *Spaceball* em *Xinput*, não há nesta um suporte total para o modelo 3003. Há somente para modelos mais recentes como o 4000 [13]. Além disto, depois de algum tempo de estudo, foram constatadas algumas tentativas frutradas em desenvolver um *driver* para o modelo 3003 [14]. Estes motivos e também a escassez de documentação para este recurso do X levaram ao abandono deste plano de trabalho e a busca por novas soluções.

5.1.2 LibSball

Após algumas pesquisas, foi encontrada uma diferente implementação que representa um diferente paradigma. Projetada por John E. Stone, a biblioteca LibSball [11] emprega uma comunicação direta com o *Spaceball* para fazer a troca de dados com o dispositivo. Mas, o que chamou a atenção foi o método empregado para integrar esta biblioteca ao *OpenGL*. Foi utilizado para isto um recurso do *Glut*, uma biblioteca que permite trabalhar com o *OpenGL* em um ambiente de janelas, que possibilita a definição de uma *callback* (rotina de atendimento a eventos) que atende sempre que o ambiente de janelas está ocioso (quando não há processamento de eventos). Sendo assim, registra-se nesta *callback* uma rotina que se comunica com o *Spaceball*, enviando e recebendo dados do mesmo, de modo que, sempre que o ambiente estiver ocioso, as ações pendentes que se referem ao *Spaceball* são processadas.

Outro ponto interessante desta biblioteca foi a parte que se refere à comunicação com o dispositivo através da porta serial. Para isto foi utilizada a biblioteca Termio [15] do C, que permite abrir e controlar

terminais os quais tem associadas portas seriais específicas. Este modelo permitiu entender o funcionamento e o controle de portas seriais no UNIX, assim como a forma de comunicação do *Spaceball*.

6 Projeto da Interface entre MTK e *Spaceball* 3003

A primeira interface desenvolvida teve como base os paradigmas estudados, em especial a biblioteca LibSball. A idéia principal foi baseada no desenvolvimento de uma estrutura de classes integradas ao MTK. Mas antes de definir esta estrutura, dois aspectos foram considerados:

1. Extensibilidade: Por ser caracterizada como uma modificação importante, a estrutura da interface poderia ser construída de forma a permitir que dispositivos semelhantes pudessem ser incluídos futuramente, dispensando o desenvolvimento de novas estruturas análogas e também alterações significativas da mesma;
2. Modularidade: A implementação da interface poderia ser feita de forma que a comunicação com o dispositivo fosse feita de uma forma bem transparente, apresentando ao usuário uma interface simples e intuitiva para se trabalhar.

6.1 Diretrizes Básicas

Foi decidido adotar o padrão de projeto (*Design Pattern*) *Abstract Factoring* [17, 18]. A implementação de um *Abstract Factory* provê uma interface para a criação de famílias de objetos relacionados ou independentes sem a necessidade da especificação de suas classes concretas. Este padrão de projeto consiste da definição dos tipos de classes: fábricas e produtos (concretos e abstratos). As classes concretas são geradas através de herança a partir das classes abstratas, que são instanciadas ao invés de seus equivalentes concretos. As fábricas concretas são classes cujos métodos tem a função de criar um produto concreto e instanciá-lo sobre a forma de um produto abstrato.

Um princípio que chamou a atenção na biblioteca *LibSball* seria também adotado, onde o evento gerado pela ociosidade do ambiente de janelas é utilizado para a leitura do estado do dispositivo, visto na seção 5.1.2. Como a classe mtkxWindow envolve diferentes gerenciadores de janelas, no caso GLUT e o GTK, os quais possuem maneiras diferentes para indicar a ociosidade do sistema de janelas, este recurso deveria ser implementado para cada gerenciador de forma independente. Assim, dada uma janela ativa qualquer criada pela mtkWin-

dow na qual o usuário trabalha, ela teria recursos próprios para saber quando o ambiente de janelas está ocioso e então fazer a leitura do estado do dispositivo concreto que realiza a interface mtkxDevice e, em seguida, ativar as *callbacks* associadas aos mesmos.

Além disto, planejava-se inicialmente a integração com o *cursor* 3D no MTK para, futuramente, estender esta integração a outros componentes, como câmeras, buscando inserir o dispositivo num contexto mais geral dentro da biblioteca. Portanto, haveria uma classe abstrata mtkxDevices e classes-filhas concretas, como mtkxDeviceSpaceball3003, que herdariam as caracterísctcas básicas de um dispositivo 3D. A classe-pai seria agregada pela classe mtkCursor, a qual poderia receber as coordenadas através destes dispositivos. A figura 3 apresenta um diagrama de classes para esta estrutura.

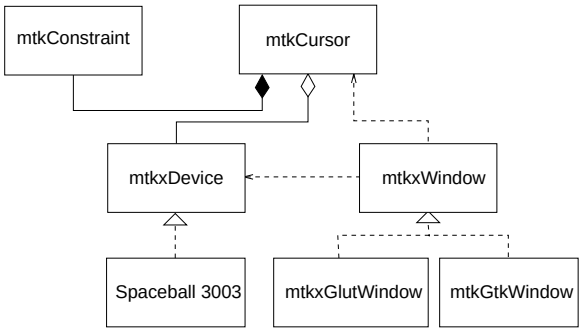

Figure 3: Projeto de interface.

Com a base do projeto definida, faltava determinar a forma com que os eventos de um determinado dispositivo seriam comunicados a um aplicativo do usuário. Para isto seria utilizado o conceito de *callbacks*. *Callbacks* são rotinas escritas dentro do aplicativo do usuário que recebem parâmetros referentes a determinados eventos de um dispositivo sempre que estes mesmos ocorrerem, tomando a seguir uma ação apropriada. No caso do *Spaceball* 3003, seriam utilizadas duas *callbacks*, uma relacionada aos eventos de movimentação da esfera e outra aos eventos de botões.

6.2 Reestruturação do MTK

O projeto de interface inicialmente proposto começou a ser implementado e, em paralelo, prosseguiam estudos referentes à integração com o MTK. Porém, foi notado que a atual estrutura do MTK não estava suficientemente preparada para receber as mudanças propostas. Por isso, antes de dar prosseguimento à implementação, foi feito um estudo para reorganizar tanto o MTK quanto a interface em questão de modo

que ambos ganhassem mais flexibilidade e compatibilidade. Isto facilitaria, portanto, o processo final de integração e conseqüentemente qualquer desenvolvimento posterior, por proporcionar uma estrutura mais coerente e legível.

O MTK foi criado em 1995, e desde então trabalharam em seu desenvolvimento diferentes pessoas. Por este motivo, alguns detalhes eram às vezes esquecidos ou mesmo ignorados. Primeiramente podemos citar a classe principal mtkCore, a qual deveria servir como interface básica para o desenvolvedor, independente da biblioteca gráfica a ser utilizada. Porém, era evidente a existência de uma utilização direta de funcionalidades específicas do OpenGL. A solução foi apresentada seguindo o conceito de padrões de projeto (*Design Pattern*), novamente com com a utilização de *Abstract Factoring* 6. Assim, a nova estrutura passou a ser contituída por uma classe abstrata (*API*) denominada mtkCore e pelas classes concretas relacionadas às bibliotecas gráficas, como por exemplo a classe mtkCoreGL que é a única atualmente desenvolvida. Construiu-se também classes responsáveis especificamente por fabricar e retornar uma estrutura completa de acordo com a biblioteca gráfica a ser utilizada. Teríamos então a classe abstrata mtkGraphLib que determina o padrão a ser seguido e as fábricas concretas como por exemplo mtkGraphLibGL que retorna um núcleo relacionado exclusivamente ao OpenGL. A implementação destas estruturas ficou a cargo do meu colega de trabalho Daniel Tost.

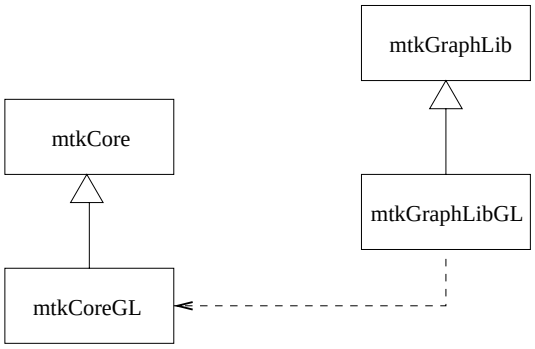

Figure 4: Nova organização da API do MTK.

Em seguida, começamos a estudar formas para dar suporte aos dispositivos não convencionais no MTK. Após analisadas as possibilidades chegamos a um interessante projeto de reestruturação, baseado em classes divididas em dois grupos principais que representariam o conjunto completo de dispositivos a ser utilizado: dispositivos 2D e 3D. Cada dispositivo pertencente a um destes grupos deveria, a princípio, ser capaz de retornar as coordenadas de sua posição em 2D (x,y) ou

em 3D (x,y,z) respectivamente, a fim de interagir com elementos específicos do MTK. Estes grupos teriam também uma classe principal responsável por propagar funcionalidades e atributos comuns a todos os dispositivos e depois, a partir destes, poderiam surgir também outras classes que representariam grupos mais específicos segundo características próprias de cada aparelho.

Desta forma, definimos uma classe principal como sendo mtkxDevice, dividida em outras duas mtkx2dDevice e mtkx3dDevice, que representam os dois grupos de dispositivos citados. No primeiro grupo estaria somente o *mouse* (classe mtkxDeviceMouse2D) e no segundo estariam o *mouse* 3D emulado (mtkxDeviceMouse3D) e o *Spaceball 3003* (mtkxDeviceSpaceball3003). Na figura 5 temos o diagrama composto por estas classes.

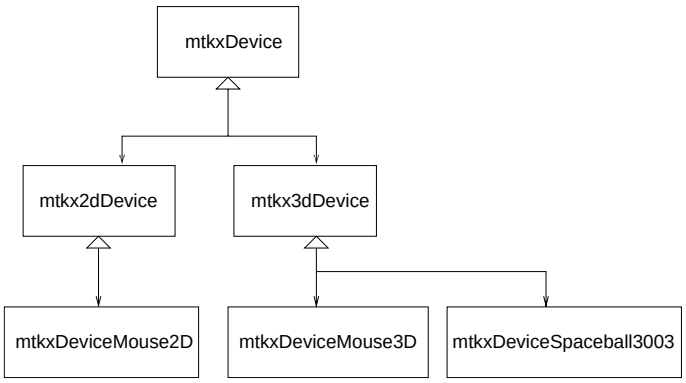

Figure 5: Nova estrutura de classes para suporte aos dispositivos.

Porém, com a evolução do projeto, vimos que haveria maior coerência se estes dispositivos manipulassem por si mesmos os atributos de determinados elementos pertencentes ao MTK, tais como o cursor e câmeras. Isto garantiria uma maior autonomia aos dispositivos e não invalidaria a divisão de classes em dois grupos. As coordenadas, por sua vez, ao invés de serem retornadas, estariam armazenadas implicitamente no elemento controlado, evitando redundâncias, já que estes valores não precisariam estar armazenados no dispositivo e no elemento ao mesmo tempo, e conseqüentemente inconsistências.

Assim, cada dispositivo trataria de forma transparente e autosuficiente seus próprios eventos, sendo capazes de processar as informações a eles condizentes e aplicá-las seguindo características e configurações específicas de cada um, tudo isso de forma trasparente para o restante do MTK.

6.3 MTKX: Interface com Sistemas de Janelas

Primeiramente procuramos situar o conjunto de classes de dispositivos no contexto do MTK. Pelo fato de os dispositivos estarem diretamente relacionados com os eventos do sistema de janelas, tais como eventos do *mouse* e de ociosidade deste sistema, chegamos a conclusão de que deveria haver alguma relação entre as estruturas de classes de ambos na elaboração da interface. Definimos, então, que deveriam haver um dispositivo 2D e outro 3D disponíveis para configuração pelo usuário e que ambos deveriam pertencer à classe referente ao gerenciador de janelas. Para isto, foram criados dois atributos privados e estáticos, um para cada tipo de dispositivo no conjunto mtkxWindow, fazendo-os existentes para qualquer janela que fosse criada com o MTKX. Os atributos privados e estáticos são aqueles que só podem ser acessados por métodos internos porém compartilhados por todas as instâncias de uma determinada classe. Construímos também métodos responsáveis por selecionar o dispositivo a ser utilizado para um tipo específico, podendo ser *mouse* 2D para dispositivos 2D e *mouse* 3D ou *Spaceball* para dispositivos 3D ou então nenhum dispositivo selecionado para qualquer um dos casos. Vale ressaltar que esta implementação pode ser expandida de forma que haja uma lista de dispositivos 2D/3D ao invés de uma simples instância permitida para cada tipo, flexibilizando a quantidade de dispositivos suportados. Porém, esta não era uma prioridade do projeto e acabou não sendo no momento implementada.

A partir daí, passamos por uma fase de reestruturação do MTKX, onde os métodos e atributos pertencentes aos dispositivos, como *mouse* 2D e 3D emulado, foram transportados e adaptados na nova implementação em progresso. Todos os métodos de mapeamento para movimentos e configuração de *Callbacks* e parâmetros foram inseridos no conjunto de seus respectivos dispositivos, inclusive as funcionalidades até então implementadas para o *Spaceball* [21], como funções específicas para a comunicação serial. Tivemos também o cuidado de adaptar os códigos já existentes tornando-os independentes das bibliotecas gráficas e dos gerenciadores de janelas que viessem a ser utilizados.

Cada dispositivo deveria ser ativado, portanto, pelos eventos do sistema de janelas os quais são obtidos a partir dos gerenciadores de janelas. Criamos então o método Activate(), pertencente a cada classe de dispositivo, responsável por indicar aos mesmos o momento em que eles devem ser ativados para processar os dados e executar as tarefas de interação. Para *mice* 2D e 3D, esta função é chamada sempre que ocorre um evento do *mouse*, obtido pelas *Callbacks* do GLUT glutMotionFunc(), glutPassiveMotionFunc() e glutMouseFunc() que se referem, respectivamente, aos eventos de movimento passivo do *mouse*,

movimento com botão pressionado e eventos de botões. Estes eventos tem os seus correspondentes no GTK: motion_notify_event, button_press_event e button_release_event.

Para o *Spaceball* é utilizada a *Callback* glutIdleFunc do GLUT e o comando gtk_idle_add() do GTK para que esta função atenda a ocorrência dos eventos de ociosidade do sistema de janelas. Um detalhe importante é que a instância do *mouse*, quando ativada, aproveita-se dos eventos que a ativam para obter os dados relacionados com a mudança de estado do dispositivo físico. Esses dados são passados para a instância no momento anterior da ativação ainda pelo MTKX, com a utilização de rotinas desenvolvidas especificamente para este fim, as quais serão vistas em 7. No caso do *Spaceball*, esses dados são obtidos internamente pelas rotinas que tratam da comunicação pela porta serial.

7 Uma Proposta de APIs

Prosseguimos com a implementação iniciamos a fase de criação das APIs, que determinam as funções disponibilizadas para o controle do dispositivo por parte de um usuário.

Após a reestruturação percebemos que apesar de serem todos dispositivos de entrada 2D ou 3D, cada um destes poderia possuir características bem peculiares as quais lhe proporcionariam funcionalidades bastante específicas ou até únicas. Desta forma, seria difícil desenvolver uma só interface que satisfizesse, de modo geral, a todos os dispositivos. A não ser que fossem implementadas funções em grande quantidade e diversidade para uma única API, a fim de abranger o maior número de possibilidades e responder a uma quantidade maior de dispositivos. Porém, normalmente haveriam funções que só teriam efeito para poucos deles. Como este tipo de implementação não nos traria benefícios, diante dos objetivos do projeto, optamos por desenvolver APIs específicas para cada dispositivo. A desvantagem disto é que esta implementação viola o encapsulamento da classe mtkxWindow, já que os atributos referentes aos dispositivos tornaram-se públicos de modo a permitir o acesso aos métodos da API dos mesmos, apesar disto não representar risco à mesma. É importante ressaltar que os atributos Device2D e Device3D, por serem do tipo mais genérico (respectivamente, classes mtkx2dDevice e mtkx3dDevice), devem sofrer um *cast* para para o tipo da classe instanciada ao serem referenciados para o acesso às APIs.

Nas seções seguintes encontram-se detalhadas as APIs para cada tipo de dispositivo atualmente disponível no MTK, de acordo com a nova estrutura de classes representada na figura 6.

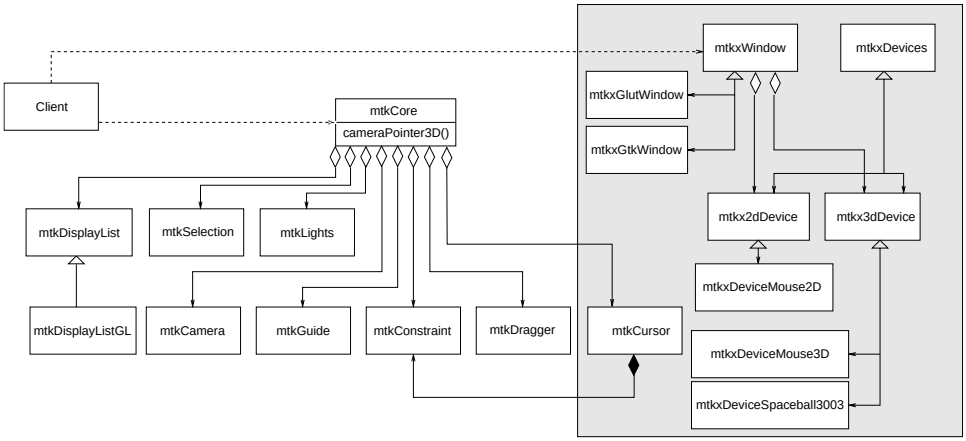

Figure 6: Novo Diagrama de Classes do MTK.

7.1 mtkxDeviceMouse2D

mtkxDeviceMouse2D() : Construtor simples. Somente inicializa os atributos da classe.

mtkxSetPosition(int x, int y) : Responsável por setar os valores para o atributo position_ que armazena a posição atual do dispositivo. É utilizada normalmente pelo MTKX ao receber eventos do *mouse*.

gmPoint2 mtkxGetPosition() : Retorna o valor do atributo position_. O valor deste atributo também é retornado pelas Callbacks de movimento. O tipo gmPoint2 é definido na biblioteca libGM, desenvolvida por Marcelo de G. Malheiros, que acompanha o código fonte do MTK [19].

mtkxSetKey(int, int) : Função que define o estado de um determinado botão do *mouse*. Esta função recebe também os valores de x e y no momento do clique (estes valores definem o *pixel* onde ocorreu o picking). Normalmente é utiliado pelo MTKX para atualizar o estado dos mesmos.

mtkxMouse2DFunc(mtkxMouse2DCallback) : Define a Callback que irá tratar eventos de botão para o *mouse*.

mtkxMotion2DFunc(mtkxMotion2DCallback) : Define a Callback que irá tratar eventos de movimento com pressionamento de botão para o *mouse*.

mtkxPassiveMotion2DFunc(mtkxPassiveMotion2DCallback) : Define a Callback que irá tratar eventos de movimento passivo para o *mouse*.

7.2 mtkxDeviceMouse3D

Conforme já emncionado na seção 6.3 um dispositivo lógico de entrada 3D pode ser ou um dispositivo físico de entrada 2D (por exemplo, *mouse* convencional) ou um dispositivo físico 3D (por exemplo, *Spaceball*). Quando se trata de um dispositivo físico 2D, MTKX emula as três coordenadas a partir das duas coordenadas de entrada e uma função de mapeamento a ser definido pelo aplicativo.

mtkxDeviceMouse3D() : Construtor simples. Somente inicializa os atributos da classe.

mtkxSetPosition(double, double, double) : Responsável por setar os valores para o atributo position_ que armazena a posição atual do dispositivo. É utilizada normalmente pelo MTKX ao receber eventos do dispositivo de entrada 3D.

mtkxSetPosition(int dx, int dy) : Responsável por setar os valores para o atributo position_ que armazena a posição atual do dispositivo. A diferença para esta função e a anterior está no fato de que nesta, os dois parâmetros se referem ao deslocamento efetuado pelo dispositivo na tela. Desta forma, estes valores são automaticamente mapeados com uma função específica que retorna o deslocamento equivalente no espaço 3D para então atualizar o atributo position_. Também é utilizada pelo MTKX para atualizar a posição do dispositivo ou receber eventos do *mouse*.

gmPoint3 mtkxGetPosition() : Retorna o valor do atributo position_. O valor deste atributo também é retornado pelas Callbacks de movimento. O tipo gmPoint2 também é definido na biblioteca libGM.

mtkxSetKey(int, int) : Função que define o estado de um determinado botão do *mouse*. Normalmente utiliado pelo MTKX para atualizar o estado dos mesmos.

mtkxSetMotionType(mtkxMotionType) : Define o tipo de mapeamento que irá emular o movimento do mouse no espaço 3D. Podendo ser MTKX_MOTION_PLANES para utilizar o chaveamento entre os planos xy e xz utilizando o botão médio do *mouse*; MTKX_MOTION_IMPLICIT para utilizar uma função implícita específica; MTKX_MOTION_CONSTRAINED para mapear o movimento sobre a superfície de um objeto e MTKX_MOTION_USER para deixar o mapeamento a cargo do usuário.

mtkxSetMotionType(mtkxMotionType, mtkId) : Equivalente a anterior porém específica para o mapeamento retrito à superfície, pois recebe como parâmetro a identificação do objeto a ser utilizado (mtkId).

mtkxSetImplicit(mtkxImplicitFunction) : Define a função implícita para mapeamento de movimento.

mtkxMappingFunc(mtkxMotionMapping) : Define a função do usuário para tratar o mapeamento de movimento.

mtkxMouse3DFunc(mtkxMouse3DCallback) : Define a Callback que irá tratar eventos de botão para o *mouse 3D*.

mtkxMotion3DFunc(mtkxMotion3DCallback) : Define a Callback que irá tratar eventos de movimento com pressionamento de botão para o *mouse 3D*.

mtkxPassiveMotion3DFunc(mtkxPassiveMotion3DCallback) : Define a Callback que irá tratar eventos de movimento passivo para o *mouse 3D*.

7.3 mtkxDeviceSpaceball3003

Para aumentar a precisão no posicionamento dos dispositivos de entrada 3D, é possível restringir (*snap*) o movimento do dispositivo sobre uma superfície ou então utilizar uma função de tratamento de dados para processá-los antes de sua utilização.

mtkxDeviceSpaceball3003() : Construtor simples. Somente inicializa os atributos da classe.

mtkxDeviceSpaceball3003(mtkxDeviceCommPort Port) : Construtor de inicialização automática. Inicializa os atributos da classe e em seguida chama a rotina de inicialização do dispositivo utilizando a porta do parâmetro Port.

mtkxDeviceSetPort(mtkxDeviceCommPort) : É utilizado para se setar a porta do dispositivo. Em seguida, o mesmo é inicializado para esta mesma porta.

mtkxDeviceIsAlive() : Método que informa sobre a disponibilidade do dispositivo, retornando *True* em caso afirmativo e *False* caso contrário.

mtkxDeviceProtocol(mtkxSpb3003Protocol) : Rotina que define o protocolo a ser adotado na comunicação, especificando as formas com que os dados serão recebidos ou transmitidos. Entretanto, obtivemos sucesso na utilização de somente um dos modos de obtenção de dados, no qual o *Spaceball* os envia periodicamente e de forma automática. Sendo assim, esta rotina ainda não possui qualquer efeito.

mtkxDeviceMode(mtkxSpb3003Mode) : Define o modo de operação para o dispositivo. O modo MTKX_SPB_CURSOR associa os

movimentos do dispositivo ao cursor 3D, MTKX_SPB_CAMERA associa à câmera e MTKX_SPB_FREE permite a utilização "livre" do dispositivo.

mtkxSetPosition(double, double, double) : Responsável por setar os valores para o atributo position_ que armazena a posição atual do dispositivo. Só é efetivo quando não está controlando nem cursor nem câmera, pois nestes casos o valor efetivo deste atributo é equivalente ao do elemento controlado.

gmPoint3 mtkxGetPosition() : Retorna o valor do atributo position_. O valor deste atributo também é retornado pelas Callbacks de movimento.

mtkxDeviceSetTranslationSens(double) : Define a sensibilidade do movimento de translação. Valores maiores resultam em um deslocamento de maior amplitude. O valor padrão é 0,1.

mtkxDeviceSetRotationSens(double) : Define a sensibilidade do movimento de rotação. Valores maiores resultam em um deslocamento angular de maior amplitude. O valor padrão é 0,1.

mtkxSetConstrain(mtkId) : Restringe o movimento do cursor à superfície do objeto identificado por mtkId.

mtkxUnsetConstrain() : Remove a restrição de movimento sobre a superfície de objetos.

mtkxSetMotionType(mtkxMotionType) : Seta o a função de tratamento para os dados obtidos do dispositivo. Diferente do método de mesmo nome para a classe mtkxDeviceMouse3D, aqui definimos uma função que irá processar os dados tridimensionais fornecidos pelo dispositivo ao invés de emular as coordenadas para um movimento 3D, com a finalidade de proporcionar um movimento mais suave ou então adequado para uma aplicação específica. Maiores detalhes serão fornecidos na seção 8.

mtkxSetNullRegion(int, int, int, int, int, int) : Define os parâmetros da função de tratamento de dados Null-Region, detalhada na seção 8.

mtkxGetNullRegion(int&, int&, int&, int&, int&, int&) : Retorna os parâmetros da função de tratamento de dados Null-Region.

mtkxControllerFunc(mtkxDeviceMotionController) : Define uma função para tratamento de dados específica, a qual recebe seis valores referentes às três translações e três rotações. Esta função poderá tratar estes valores e depois retorná-los, também detalhada na seção 8.

mtkxKeyFunc(mtkxDeviceKeyCallback) : Define a Callback que irá tratar eventos de botão para o *Spaceball*.

mtkxMotionFunc(mtkxDeviceMotionCallback) : Define a Callback que irá tratar eventos de movimento com botão pressionado para o *Spaceball*.

mtkxPassiveMotionFunc(mtkxDevicePassiveMotionCallback) : Define a Callback que irá tratar eventos de movimento passivo para o *Spaceball*.

Os métodos privados da classe mtkxDeviceSpaceball3003 estão incluídos no Apêndice B.

8 Resultados

Para validar e depurar as APIs desenvolvidas, foram feitos testes de interações 3D utilizando o *Spaceball 3003* e as funções de *picking* e *snapping* desenvolvidas por Tost [20]. Para isto, foram desenvolvidos dois aplicativos que exploram estes recursos, ambos detalhados a seguir.

8.1 *Picking* 3D com *Spaceball*

O algoritmo de *picking* 3D desenvolvido por Tost faz um uso inteligente de algumas funcionalidades providas pelo *OpenGL* de modo a estender o conceito de seleção de objetos para ambientes interativos 3D. Neste exemplo, a idéia é agregar esta nova funcionalidade ao *Spaceball* e efetuar testes com os paradigmas desenvolvidos. Para a integração, fizemos com que a rotina de seleção 3D fosse invocada automaticamente ao se pressionar o botão esquerdo do *Spaceball*, na ausência de alguma função definida para atender eventos de pressionamento de botão.

O aplicativo permite que o usuário interaja com alguns cubos sólidos, posicionados randomicamente na cena, cada um podendo ser selecionado e movido até um segundo cubo em *wireframe*, de mesma cor, até que o par tenha suas posições relativamente próximas. O objetivo é atingido quando todos os cubos sólidos estiverem dentro dos seus cubos em *wireframe* correspondentes.

Esta implementação ajudou a detectar algumas dificuldades provenientes da utilização do *Spaceball* para interação em ambientes 3D. Elas estão diretamente relacionadas com a grande liberdade de movimentos proporcionada pelo dispositivo, o que exige do usuário tempo significativo para adaptação. Algumas funcionalidades adicionais foram sugeridas a fim de minimizar este tempo de adaptação. São elas:

Região Nula : Este algoritmo, conhecido como *Null-Region* [11], atua como um filtro que ignora baixos valores para cada componente dos dados de movimento obtidos. Desta forma, é determinada uma região nula, onde qualquer movimento do dispositivo, abaixo de um determinado limiar, é ignorado. O próposito é tentar rejeitar os chamados *movimentos espúrios*, movimentos provocados de forma não intencional, os quais normalmente possuem valores de baixa magnitude. Um exemplo prático ocorre quando tenta-se mover o *Spaceball* ao longo de um único eixo, em que até uma pequena tremulação pode afetar o resultado final.

Função de Controle : Uma outra solução seria flexibilizar a interface construída de modo a permitir ao usuário que definisse sua própria função para processamento dos dados do dispositivo. Desta forma, poderia ser aplicada aos valores obtidos para cada componente da translação ou rotação uma função qualquer cabível, como uma função senoidal ou um polinômio atenuador. Desta forma, o usuário tem o controle total sobre o movimento do *Spaceball*. Neste exemplo, foi implementada uma função exponencial do tipo f(v) = A*v*exp(B*v), onde v é o valor de uma das componentes de movimento. Assim, proporcionamos um movimento com sensibilidade variável, mais precisos para deslocamentos menores e mais rápidos para deslocamentos maiores.

Estas funcionalidades podem ser habilitadas utilizando-se o método mtkxSetMotionType() da classe *Spaceball* com os parâmetros:

MTKX_MOTION_NULLREGION: para habilitar o processamento dos dados de movimento pelo algoritmo de *Null-Region*;

MTKX_MOTION_CONTROLLER: para utilizar uma função definida pelo usuário através do método mtkxControllerFunc();

MTKX_MOTION_FREE: para não utilizar nenhum destes algoritmos.

Outro problema detectado a partir deste aplicativo foi inexistência de uma especificação que relacionasse o dispositivo com uma determinada janela quando mais de uma fosse utilizada simultâneamente. Sendo assim, cada gerenciador de janelas atuava indiretamente na escolha da janela de foco, a partir de suas próprias definições de janela ativa. Isso foi resolvido com a criação de uma especificação própria para o MTK e com a utilização de eventos do *mouse*. Definimos então que só poderia existir uma única janela de foco, a qual seria responsável pelo gerenciamento dos eventos relacionados aos dispositivos, sendo que uma determinada janela só receberia o foco se algum evento de *mouse* ocorresse diante da mesma.

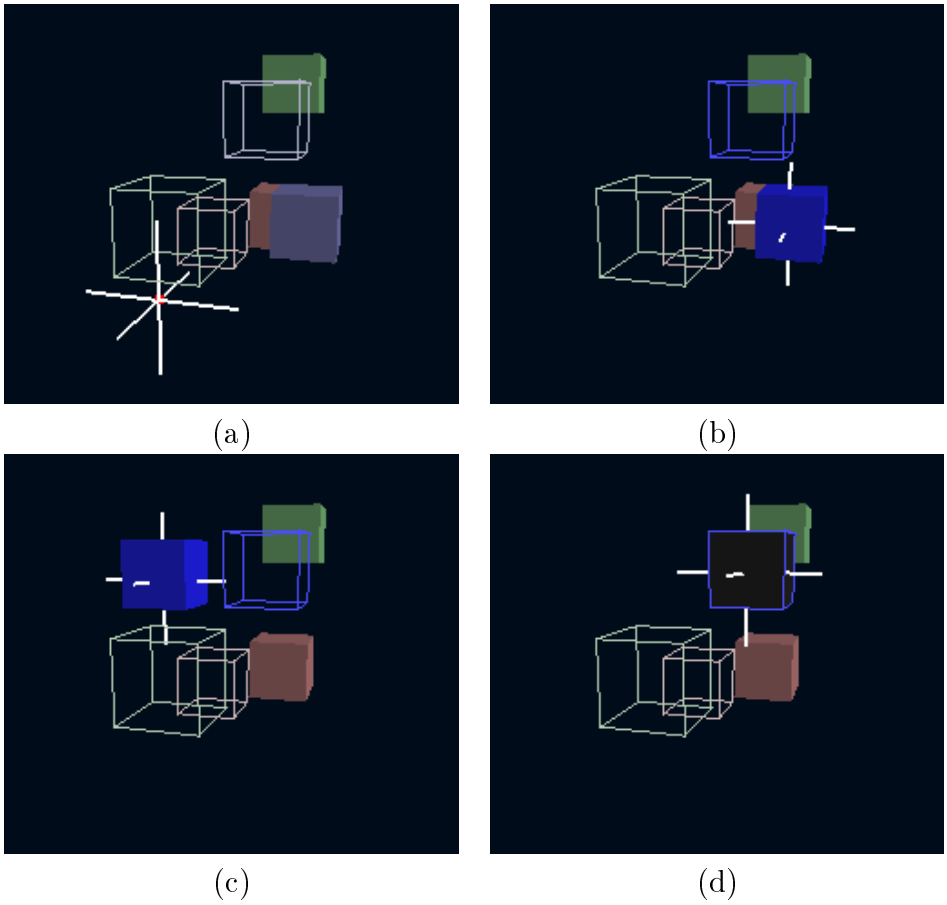

Figure 7: Aplicativo para seleção 3D.

8.2 *Snapping* 3D com *Spaceball*

Dando prosseguimento à incorporação de funcionalidades ao *Spaceball* tiramos proveito do algoritmo de *snapping* 3D, uma outra implementação de Tost [20]. A técnica de *snapping* 3D consiste em fazer com que o *cursor* 3D se mova interativamente sobre uma superfície convexa suave. Para isto fizemos algumas adaptações na implementação feita para o *mouse* [20], adequando-a ao *Spaceball*.

Definimos que um deslocamento no dispositivo nos eixos X e Z provocaria um movimento sobre a superfície enquanto o deslocamento no eixo Y levaria a um movimento sobre o vetor normal desta superfície, funcionalidade exclusiva dos dispositivos 3D.

Os aplicativos desenvolvidos para testar a implementação com o *mouse* foram utilizados para teste, porém contando agora com o *Spaceball*. Tost desenvolveu aplicativos que implementam o *snapping* sobre uma esfera e um cubo. Os testes realizados mostram que esta técnica

é valida também para os dispositivos 3D e que estes introduzem possibilidades novas e interessantes de movimento com a adição de mais um grau de liberdade, como por exemplo um movimento orbital ao redor de uma esfera.

Porém, estes aplicativos mostram que este recurso poderia evoluir ainda mais se fossem considerados os movimentos de rotação permitidos pelo dispositivo. Estes movimentos poderiam servir, por exemplo, para alterar a direção de um caminho em curso sobre uma superfície. Contudo, um estudo mais cauteloso deveria ser executado antes de introduzir estas idéias ao modelo atual.

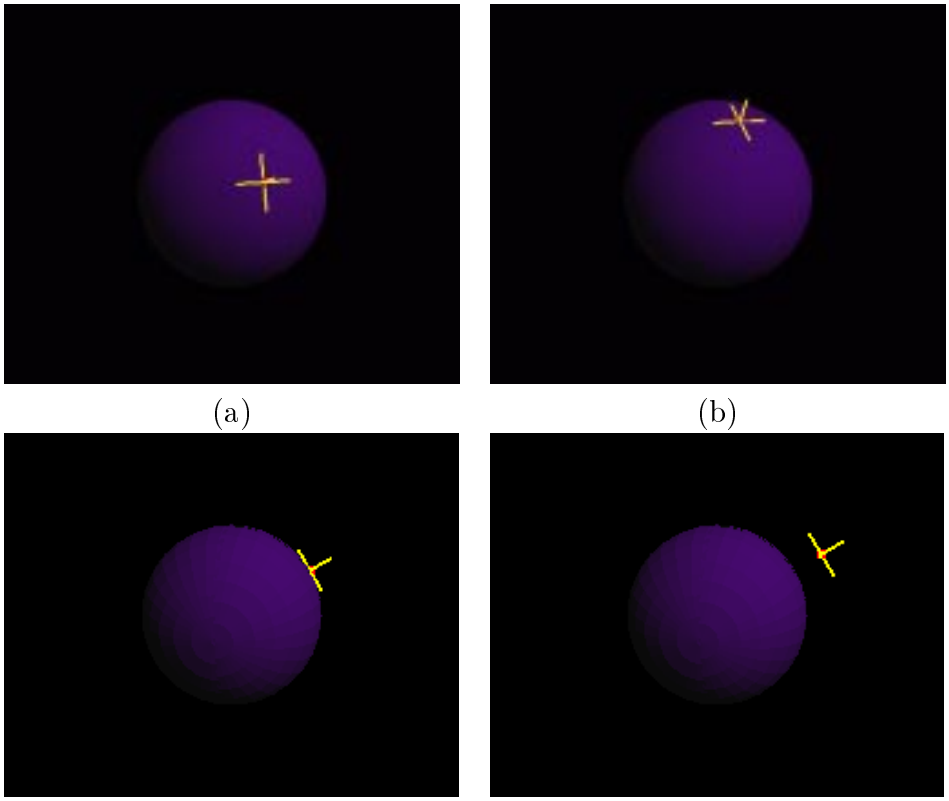

Figure 8: Movimento restrito à superficie de uma esfera.

A figura 8 mostra duas seqüências de movimento para esta aplicação, a primeira sobre a superfície de uma esfera (itens `a` e `b` para movimento no eixo XZ do *Spaceball* e `c` e `d` para o eixo Y normal à superfície) e a segunda sobre um cubo (itens `e` e `f`).

(e) (f)

Figure 9: Movimento restrito à superficie de um cubo.

9 Avaliação Comparativa entre Mouse Emulado e *Spaceball*

Concluídas a interface e a integração com o MTK, seria interessante avaliar a efetividade dos dispositivos implementados e comparando, quando possível, o movimento proporcionado por uma entrada 3D real com a entrada emulada.

Utilizando o aplicativo de *picking* 3D de 8.1 e contando com alguns colegas de trabalho como voluntários, realizamos um teste bastante informal onde o objetivo era avaliar os modelos propostos, tomando-se os tempos para execução de determinadas tarefas e recebendo comentários e sugestões.

O teste foi subdividido em 7 etapas. Na primeira etapa medimos o tempo para o usuário colocar cada um dos 3 cubos da cena em lugares específicos utilizando o *mouse* 3D emulado. Nas próximas 3 etapas a ação cronometrada foi a mesma, porém utilizando o *Spaceball*, respectivamente, nos modos de operação padrão do dispositivo, com processamendo de região nula e utilizando-se uma função de controle exponencial como em 8.1. As próximas 3 etapas foram similares, diferindo apenas no modo de seleção dos objetos, onde foi permitida a utilização simultânea do *Spaceball* e do *mouse* 2D apenas para este propósito. Antes de cada teste, foi permitido ao usuário testar os dispositivos para que compreendessem melhor suas reações e funcionalidades. A tabela seguinte apresenta os resultados obtidos:

Testes/Tempos	Mínimo (s)	Médio (s)	Máximo (s)
Mouse 3D Emulado	15	44	87
Spaceball modo Normal	16	48	90
Spaceball modo Null Region	16	51	132
Spaceball com controle exp.	14	34	63
Spb. e Mouse modo Normal	10	25	39
Spb. e Mouse modo NR.	13	35	73
Spb. e Mouse com ctrl. exp.	7	22	50

Vale ressaltar que fizeram os testes tanto pessoas com bons conhecimentos no uso de computadores e/ou da computação gráfica (aproximadamente 25%), com conhecimentos intermediários (aproximadamente 15%) e com poucos conhecimentos (60%). Outro fator considerado no teste foi a necessidade de se posicionar os cubos aleatóriamente para a execução consecutiva das sete etapas descritas, porém, estas se repetiam para cada usuário garantindo a eles igualdade de condições. Os resultados acima refletem os comentários feitos pelos usuários e nos possibilitaram fazer o levantamento dos aspectos favoráveis e desfavoráveis para cada dispositivo:

Mouse 3D : Muitos dos usuários tiveram facilidade em se habituar ao movimento do cursor 3D controlado pelo *mouse* em função de uma familiaridade já existente com o dispositivo. Porém, muitos citaram que, ao contrário do *Spaceball*, ele perde em flexibilidade por exigir o chaveamento dos planos de movimentação através de um dos seus botões, tornando os movimentos mais demorados e complicados, principalmente no plano XZ. Um comentário relevante relata sobre a perda de recursos que esta implementação pode trazer devido a utilização de um dos botões do mouse para o chaveamento dos planos e outro para efetuar o *picking*, podendo não ser viável para dispositivos com apenas 2 botões. Isto explica o fato de a média de tempos ter sido razoável.

Spaceball : Para este dispositivo, muitos usuários tiveram certa dificuldade no manuseio, o que é normal por ser um dispositivo pouco convencional. Após algumas tentativas, as reações passavam a fazer sentido e os tempos já podiam ser tomados. Muitos destacaram que com o hábito, o *Spaceball* tornava-se mais intuitivo para o ambiente 3D e tirava certas vantagens por contar com alguns recursos exclusivos, como a movimentação da câmera. Os tipos de movimento permitidos também foram avaliados, apontando para a função de controle exponencial como a mais apropriada para o teste em questão, explicando os bons resultados obtidos com esta configuração. A utilização simultânea dos dispositivos foi também considerada interessante, pois combinava a

facilidade da seleção convencional com a flexibilidade do *Spaceball*, o que proporcionou os bons resultados obtidos. A seleção 2D convencional, apesar de se mostrar bastante objetiva para o este aplicativo e por ser intuitiva para usuários que já possuiem alguma familiaridade com ambientes gráficos, perde em termos de disponibilidade de recursos para a seleção 3D desenvolvida por Tost, por isso, apesar dos resultados, não podemos afirmar que um tipo de seleção é melhor que outro, apenas que cada um será mais apropriado para um caso específico. Isto fortalece a idéia de que ambos paradigmas são válidos para fins de pesquisa e desenvolvimento.

10 Conclusões

A análise de todo o trabalho desenvolvido nos permite concluir que os objetivos almejados foram de certa forma alcançados, incluindo basicamente todos os itens descritos na seção 2. As experiências obtidas vão desde o estudo de linguagens de programação até a implementação de interfaces com dispositivos, formando uma base de conhecimentos considerável.

O MTK conta atualmente com uma estrutura que, além de implementar a interface com o *Spaceball*, o prontifica para a recepção de modelos diversos de dispositivos de entrada. Além disto, esta interface apresenta funcionalidades inéditas na biblioteca que tornam o *Spaceball* uma ferramenta mais eficiente e versátil. Os testes ajudaram a avaliar a implementação do projeto como um todo, permitindo-nos concluir que os resultados obtidos são satisfatórios.

Ainda foram realizadas tarefas como o início da documentação da API do MTK e a reestruturação desta biblioteca, que visam facilitar futuras modificações por parte dos desenvolvedores e tornar o uso da biblioteca mais fácil para os programadores.

Durante o período de desenvolvimento do trabalho, decidimos pela confecção de um artigo; em co-autoria com Wu, Tost e Batagelo; submetido como contribuição para o Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens (Sibgrapi). Existe ainda a intenção da inclusão deste trabalho no Workshop de IC do Sibgrapi 2003.

Por fim, tivemos a integração com as funcionalidades desenvolvidas por Tost em seu projeto de pesquisa, que permitiram combinar a interação com dispositivos 3D com os interessantes algoritmos de *picking* e *snapping* 3D, extendendo ainda mais a gama de recursos disponibilizados pelo MTK.

References

[1] 3Dconnexion. Articles - `http://www.logicad3d.com/articles`.

[2] 3Dconnexion. How Two-Handed Control Reduces The Time and Cost of 3D Projects And Results in Better Designs - `http://www.logicad3d.com/articles/benefits.pdf`.

[3] 3Dconnexion. Navigating in the third dimension - `http://www.logicad3d.com/articles/2002-01_Navigation_3rd_dimension.pdf`.

[4] Immersion. 3D interaction overview - `http://www.immersion.com/products/3d/interaction/overview.shtml`.

[5] Interfaces hápticas - `http://haptic.mech.nwu.edu/`

[6] Discreet. Discreet Products - 3ds Max - `http://www.discreet.com/products/3dsmax/`.

[7] Alias—Wavefront. About Maya - `http://www.aliaswavefront.com/en/products/maya/index.shtml`.

[8] Newtek. Lightwave 3D - Newtek - `http://www.newtek.com/products/lightwave/index.php`.

[9] Russell Turner, Francis Balaguer, Enrico Gobbetti, Daniel Thalmann. Physically-based interactive camera motion control using 3D input devices - `http://ligwww.epfl.ch/~thalmann/papers.dir/CGI91.pdf`.

[10] *Spaceball* and Solid Edge - `http://www.jsquaredmech.com/hardware/labtec2.htm`.

[11] Stone, John E. LibSball - `http://jedi.ks.uiuc.edu/~johns/projects/libsball`.

[12] Taylor, Owen. The XInput HowTo - `http://www.gtk.org/~otaylor/xinput/howto/index.html`.

[13] Krahn, Joe. Resources for XInput drivers and 6D controllers - `http://www.geocities.com/joekrahn/`.

[14] Miranda, Javier Abadia. *Spaceball 3003* - XEvents - `http://oss.sgi.com/projects/performer/mail/info-performer/perf-98-01/0257.html`.

[15] Termio - General Terminal Interface - UNIX MANUAL PAGE.

[16] Mesquita, Leonardo A. G. de. Relatório do Projeto de Cursores 3D com uso de dispositivos 2D. Março de 2001. Processo N.º 00/09619-3.

[17] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Design Patterns; Elements of Reusable Object-Oriented Software. (Addison-Wesley Professional Computing) ISBN: 0201633612, 1995.

[18] Data & Object Factory - Developer Training Home Page - `http://www.dofactory.com/patterns/patterns.aspx`

[19] MTK - Manipulation Tool Kit Home Page. `http://www.dca.fee.unicamp.br/~ting/Projects/mtk`.

[20] Tost, Daniel. Relatório Final do Projeto de Pesquisa - Extensão das Funções de Emulação do Cursor3D no MTK. Fevereiro de 2003. Processo Fapesp N.^{o} 02/01162-0.

[21] Abrantes, Marcel. Desenvolvimento de uma Interface entre MTK e *Spaceball*. Relatório Parcial de Projeto de Pesquisa de Iniciação Científica FAPESP. Processo n^{o}02/01161-3. Novembro 2002.

A Protocolo de Comunicação do *Spaceball* 3003

Neste apêndice se encontram informações básicas sobre o protocolo de comunicação do *Spacetek Spaceball* 3003. O dispositivo se comunica através de uma interface do tipo RS-232 a 9600bps, com 8 *bits* de dados, sem paridade e 1 *bit* de parada. Os pacotes de dados são constituídos por um *header* de um *byte* no formato ASCII, zero ou mais *bytes* de dados e um *Carriage Return* (Hexadecimal 0D) undicando o final do pacote. Os pacotes mais importantes seguem descritos nas subseções abaixo.

A.1 Pacote de Eco

Um pacote com o caractere "%" no *header* será ecoado de volta com o símbolo "%" substituído por um caractere "espaço". Isto serve principalmente para detectar a presença do dispositivo.

A.2 Pacote de Reset

Para resetar o *Spaceball* deve-se enviar um pacote contendo a *string* a seguir no padrão ASCII:
'@RESET'
O dispositivo irá executar um *reset* interno e em seguida devolver as seguintes *strings*:

@1 Spaceball Alive and well after a reset.
@2 Firmware version x.xx created on dd.mmm.yy,
onde x.xx é a versão do firmware e dd.mmm.yy é a data de criação do mesmo.

Durante o *reset* o dispositivo emite dois *beeps*, seta o Modo de Comunicação para Binário (Padrão) e define a posição atual da esfera como ponto de referência inicial.

Este pacote também pode ser utilizado para se identificar o dispositivo.

A.3 Pacote de Beep

O pacote de dados para executar um *Beep* sonoro no *Spaceball* consiste de:
'B[sequência de bytes]',

onde cada byte pode ser:
'a' igual a 1/30 segundos de *beep*;
'o' igual a 15/30 segundos de *beep*;
'A' igual a 1/30 segundos de pausa;
'O' igual a 15/30 segundos de pausa;

Estes caracteres são colocados em uma fila para o processamento, com um limite de 15 caracteres (bytes).

A.4 Pacote do Modo de Comunicação

Para se setar o modo de comunicação deve-se enviar um pacote com o seguinte formato:
'Cm',

onde m pode ser:
'B' modo de comunicação binário com terminador de pacote CR;
'b' mode de comunicação binário com terminador de pacote CRLF;
'P' modo ASCII com terminador CR;
'p' modo ASCII com terminador CRLF;
'CR' equivale ao *Carriage Return* (Hexadecimal 0D) e 'LF' equivale ao *Line Feed* (Hexadecimal 0A).

A.5 Pacote de Movimento

O pacote de movimento tem o seguinte formato:
'DppxxyyzzXXYYZZ',

onde
pp é o período;
xx,yy,zz são os componentes do vetor de translação;
XX,YY,ZZ são os componentes do vetor de rotação.

A.6 Pacote de Erros

Se ocorrer algum erro, o *Spaceball* irá gerar um pacote de erro da seguinte forma:
'E[código dos erros]'

Em [código dos erros] podem vir caracteres sempre maiúsculos indicando a ocorrência de um ou mais erros. Esses caracteres são:
'@' Erro de Hardware
'A' Erro de *Checksum* ao calibrar
'B' Erro de Hardware
'C' Erro de Hardware
'D' *Timeout* ao transmitir
'E' *Overflow* ao receber
'F' Erro de recepção
'G' Erro na fila de *Beep*
'H' Pacote muito longo

A.7 Pacote de Ajuda

Existem dois tipos de pacotes de ajuda, um para informação sobre versão de *firmware* e outro para informações sobre os limites de sensibilidade.

Para requisitar a versão do *firmware*:
'hv'
O dispositivo responde:
'HvVx.xx dd-mmm-yy,

onde x.xx é a versão do firmware e dd-mmm-yy a sua data de criação.

Para requisitar os limites de sensibilidade:
'hs'
O dispositivo responde:
'Hssff.ffN t.ttttNm nnbit',

onde:
ff.ff é um valor referente a força em Newtons (N)
t.ttt é um valor referente ao torque em Newton-metros (Nm)
nn é um valor referente à quantidade de bits de resolução
A força e o torque correspondem ao máximo valor que cada componente x,y ou z pode ser gerar.

A.8 Pacote dos Botões

Quando um botão é pressionado no *Spaceball* um pacote é gerado com o seguinte formato:
'Kbb';

onde 'bb' são dois *bytes* da forma '01ZR0000' e '01LR0000' e:
Z é o botão de *Rezero*,
L é o botão esquerdo,
R é o botão direito.
O estado dos botões pode ser consultado a qualquer momento enviando um pacote com o conteúdo 'k'. Isto vale para outros pacotes, lembrando que deve-se usar letras minúsculas para requisitar uma informação e que as respostas vem sempre com letras maiúsculas.

A.9 Pacote de Troca de Modo

Para fazer com que o Spaceball envie automaticamente os dados de movimento deve-se usar o seguinte comando:
'M'
Tendo o modo como parâmetro:
'SS',
onde 'SS' é o modo ao qual se quer se quer setar o *Spaceball*. Exemplo 'MSS', que faz com que o *Spaceball* envie automaticamente um pacote de Movimento 'M' quando a esfera for manipulada.

A.10 Pacote de Rezero

O *Spaceball* percebe as forças e torques sempre a partir de uma posição relativa inicial, denominada posição de descanso (*Rezero*). Existem três momentos em que esta posição é definida: quando o dispositivo é ligado, quando é pressionado o botão *Rezero* e quando é enviado um pacote de *Rezero* 'Z' vazio.

Para isto, deve-se enviar um pacote com o seguinte conteúdo:
'Z'

O *Spaceball* não irá responder com um pacote de dados de *Rezero*. Para recebê-lo, deve se fazer a seguinte requisição:
'z'
O *Spaceball* retornará um pacote com o seguinte formato:
'Z[posição de zero]',
onde [posição de zero] possui 12 *bytes* de informação não especificados, devendo ser utilizado por programas de diagnóstico.

B Classe mtkxDeviceSpaceball3003

Neste apêndice temos uma breve descrição dos métodos internos desenvolvidos para o *Spaceball 3003*.

Activate() : Método responsável por "ativar" o dispositivo, fazendo com que o mesmo obtenha os dados pela porta serial, processe os dados obtidos e execute o controle sobre o elemento especificado pelo parâmetro mode_ setado por mtkxDeviceMode().

bool mtkxDeviceCommOpen_() : É responsável por setar as propriedades da porta a ser utilizada, caso esta esteja definida no atributo comm_port, e em seguida abrir a mesma para permitir a comunicação. No UNIX, uma porta serial está associada a um terminal que pode ser aberto como um arquivo de dados comum, o qual atua como um *streaming* de informações. A biblioteca Termio [15] em conjunto com algumas outras bibliotecas padrão do C permitem a manipulação destes arquivos. Caso um arquivo seja aberto corretamente, é associado a ele um descritor (atributo *file_desc_*) e o método retorna True, caso contrário *file_desc_* é setado como NULO (*NULL*) e o método retorna False.

int mtkxDeviceCommWrite_(char * buffer) : Envia todo o conteúdo do parâmetro buffer_ para o dispositivo. Este método retorna um inteiro que representa a quantidade de caracteres efetivamente enviados.

int mtkxDeviceCommRead_() : Transfere todo o conteúdo do buffer de dados do dispositivo para a variável interna buffer_. Este método retorna um inteiro que representa a quantidade de caracteres lidos.

bool mtkxDeviceCommClose_() : Fecha a porta encerrando as comunicações. Este método retorna True se a porta for fechada e False em caso de erro.

mtkxDeviceInit_() : Rotina de inicialização padrão para dispositivos. Testa a presença do dispositivo e em seguida inicializa o mesmo.

void mtkxMappingConstraint_() : Função que retorna um ponto result resultado do mapeamento do movimento amarrado a um plano. Para os parâmetros passados à função temos os vetores dx e dy, que definem o plano e a direção do movimento; o ponto p é o ponto inicial da operação e, por fim, os valores x e y que correspondem ao deslocamento efetuado nas direções x e y. Este método é equivalente ao desenvolvido para o mouse 3D emulado [20].